# 1 Preliminaries

## 1.1 Constrained Horn Clauses

**Definition 1.** A Constrained Horn Clause is a formula

$$C \wedge B_1 \wedge \ldots \wedge B_n \implies H$$

where

- $C$ is a constraint over the background theory
- each $B_i$ is an uninterpreted predicate applied to (interpreted) terms over the background theory
- $H$ is, similarly, an uninterpreted predicate applied to interpreted terms, or it is the constant $\bot$.

$H$ is called the *head* of the clause and $C \wedge B_1 \wedge \ldots \wedge B_n$ is called the *body*. Equivalently, we can write CHC in a Prolog format

$$H :- C \wedge B_1 \wedge \ldots \wedge B_n$$

A set of CHCs is (syntactically) *solvable* if there exists an interpretation of the uninterpreted predicates in the background theory such that each CHC yields a valid formula when the uninterpreted predicates are substituted by their interpretation.

**Example 1.** Consider the following system of CHCs with the background theory of integer linear arithmetic.

$$x = 0 \wedge y = 0 \implies Inv(x, y) \tag{1}$$
$$Inv(x, y) \wedge x' = x + y \wedge y' = y + 1 \implies Inv(x', y') \tag{2}$$
$$Inv(x, y) \wedge x < 0 \implies \bot \tag{3}$$

The solution for this set of CHCs is the interpretation $x \geq 0 \wedge y \geq 0$ for $Inv(x, y)$.

## 1.2 Hierarchical planning

Hierarchical planning, or *Hierarchical Task Networks* (HTN) planning in full, uses the language of first-order logic with the following sets: $T$ is a set of *type symbols*, $V$ is a set of *typed variable symbols*, $P$ is a set of predicate symbols and $C$ is a set of typed constants. There are two distinct kinds of tasks: *primitive* tasks (also called *actions* and *compound* tasks (also called *abstract* tasks).

**Definition 2** (TaskNetwork)**.** A *task network tn* over a set of task names $X$ (first-order atoms) is a tuple $(I, \prec, \alpha, VC)$ with the following elements:

1. $I$ is a (possibly empty) set of task identifiers.
2. $\prec$ is a strict partial order over $I$.
3. $\alpha : I \to X$ maps task identifiers to task names.
4. $VC$ is a set of variable constraints.

Each constraint can bind two task parameters to be (non-)equal and it can constrain a task parameter to be (non-)equal to a constant, or to (not) be of a certain type.

A task network is *ground* if all parameters are bound to (or replaced by) constants from C.

An *action* a is a tuple $(name, pre, eff)$, where *name* is its task name, a first-order atom consisting of the (actual) name followed by a list of typed parameter variables. *pre* is its *precondition*, a first-order formula over literals over predicates from $P$. *eff* is its effect, a conjunction of literals over predicates from $P$. We define $eff+$ and $eff-$ as the sets of atoms occurring non-negated/negated in *eff*. All variables used in *pre* and *eff* have to be parameters of *name*. We also write $name(a)$, $pre(a)$, and $eff(a)$ to refer to these elements. We require that action names $name(a)$ are unique.

A *compound task* is simply a task name, i.e., an atom. Its purpose is not to induce state transition, but to reference a pre-defined mapping to one or more task networks by which that compound task can be refined. This mapping is given by a set of (decomposition) methods $M$. A method $m \in M$ is a triple $(c, tn, VC)$ of a compound task name $c$, a task network $tn$, and a set of variable constraints $VC$ that allow to (co)designate parameters of $c$ and $tn$.

**Definition 3** (Planning Domain)**.** A planning domain $D$ is a tuple $(L, TP, TC, M)$ defined as follows:

- $L$ is the underlying predicate logic.
- $TP$ and $TC$ are sets of primitive and compound tasks.
- $M$ is a set of decomposition methods with compound tasks from $TC$ and task networks over the names $TP \cup TC$.

The domain implicitly defines the set of all states $S$, being defined over all subsets of all ground predicates.

**Definition 4** (Planning Problem)**.** A planning problem $P$ is a tuple $(D, s_I, tn_I, g)$, where:

- $D$ is a planning domain.
- $s_I \in S$ is the initial state, a ground conjunction of positive literals over the predicates.
- $tn_I$ is the initial task network (not necessarily ground).
- $g$ is the goal description, a first-order formula over the predicates (not necessarily ground).

# 2 Encoding hierarchical planning problems into Constrained Horn Clauses

To encode a HTN planning problem $P$ into a set of CHCs, we follow these steps:

1. We introduce a boolean variable for each ground fact, i.e., for each instantiation of each predicate in our language with constants of appropriate types for the arguments. These variables represent the state of the world at a given state. If the variable is assigned to true, the ground fact holds in the current state, if the variable is assigned to false, the fact does *not* hold in the current state. We also use primed version of the variables to represent the variables in the *next* state.

2. For each task (both abstract and primitive) $t(x_1, \ldots, x_n)$ we define an uninterpreted predicate $t(i_1, \ldots, i_n, \mathbf{s}, \mathbf{s}')$ where $i_1, \ldots, i_n$ are variables of sort Integer, $\mathbf{s}$ are the boolean variables introduced in step 1 and $\mathbf{s}'$ are the primed version of $\mathbf{s}$. The intuition behind these uninterpreted predicates is the following: $t(i_1, \ldots, i_n, \mathbf{s}, \mathbf{s}')$ is true when executing the abstract task $t$ on arguments defined by $i_1, \ldots, i_n$ changes the state from $\mathbf{s}$ to $\mathbf{s}'$. $i_j$ for $1 \leq j \leq n$, picks the $i_j$-th constant from the domain of the type of $x_j$.

3. For each abstract task $t$ and each of its possible decomposition methods $m : t \to \langle d_1, \ldots, d_m \rangle$ (we assume the *total* order, so the decomposition tasks form a *sequence*) we introduce a clause of the following form:

$$d_1(args_{d_1}, \mathbf{s}^{(0)}, \mathbf{s}^{(2)}) \wedge \ldots \wedge d_m(args_{d_m}, \mathbf{s}^{(m-1)}, \mathbf{s}^m) \wedge Const_{args} \implies t(args_t, \mathbf{s}^{(0)}, \mathbf{s}^m)$$

where $Const_{args}$ ensures that the arguments $args_{d_1}, \ldots, args_{d_m}, args_t$ respect the constraints from the definition of the decomposition method.

4. For each action (primitive task) we introduce a number of clauses, each corresponding to one ground instances of the action.

$$\bigwedge_{p \in pre(a)} \wedge \bigwedge_{e \in eff(a)} \implies a(args_a, \mathbf{s}, \mathbf{s}')$$

where preconditions are constraints on the current state $\mathbf{s}$ and effects are constraints on the next state $\mathbf{s}'$; $args_a$ are numbers enumerating the set of constants of the corresponding argument's type. The set of (boolean) facts in preconditions and effects are given by the concrete constants present in the ground instance of the action.

5. Finally, we introduce a clause defining the initial state $s_I$ and the initial sequence of tasks $tn_I$.

$$\bigwedge_{s \in s_I} s \wedge \bigwedge_{s \notin s_I} \neg s \bigwedge_{t_i \in tn_I} t_i(args_{t_i}, \mathbf{s}^{(i-1)}, \mathbf{s}^{(i)}) \implies \bot$$

This clause encodes the question if the given initial task sequence can be executed from the initial state.

The resulting set of CHCs is satisfiable if and only if there exists no solution to the planning problem. Moreover, if the CHCs problem us unsatisfiable, then the solution for the planning problem can be extracted from the counter-example.

**Example 2.** Consider this simple transportation planning problem (using the HPPL language):

```
(define (domain domain_htn)
  (:types
    location - object
    vehicle - locatable
    locatable - object
  )
  (:predicates
    (road ?arg0 - location ?arg1 - location)
    (at ?arg0 - locatable ?arg1 - location)
  )

  (:task get_to
    :parameters (?v - vehicle ?l - location)
  )

  (:method m_drive_to_ordering_0
    :parameters (?l1 - location ?l2 - location ?v - vehicle)
    :task (get_to ?v ?l2)
    :subtasks (task0 (drive ?v ?l1 ?l2)
    )
  )

  (:method m_drive_to_via_ordering_0
    :parameters (?l2 - location ?l3 - location ?v - vehicle)
    :task (get_to ?v ?l3)
    :subtasks (and
     (task0 (get_to ?v ?l2))
     (task1 (drive ?v ?l2 ?l3))
```

```
    )
  )

  (:method m_i_am_there_ordering_0
    :parameters (?l - location ?v - vehicle)
    :task (get_to ?v ?l)
    :subtasks (and
     (task0 (noop ?v ?l))
    )
  )

  (:action drive
    :parameters (?v - vehicle ?l1 - location ?l2 - location)
    :precondition
      (and
        (at ?v ?l1)
        (road ?l1 ?l2)
      )
    :effect
      (and
        (not (at ?v ?l1))
        (at ?v ?l2)
      )
  )

  (:action noop
    :parameters (?v - vehicle ?l2 - location)
    :precondition
      (and
        (at ?v ?l2)
      )
    :effect ()
  )
)

(define
  (problem pfile01)
  (:domain  domain_htn)
  (:objects
    c0 - location
    c1 - location
    t0 - vehicle
  )
  (:htn
    :parameters ()
    :subtasks (task0 (get_to t0 c1)
```

```
  )
)
(:init
  (road c0 c1)
  (road c1 c0)
  (at t0 c0)
)
```

Using the translation we obtain

- The boolean variables for all the ground instances of predicates:

$$r(c0,c1), r(c1,c0), r(c0,c0), r(c1,c1), at(t0,c0), at(t0,c1)$$

- Horn clauses for the abstract tasks:

$$\begin{aligned} &getto(v,l,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\ &\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)})\\ &:- drive(v,il,l,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\ &\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)}) \end{aligned}$$

$$\begin{aligned} &getto(v,l,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\ &\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)})\\ &:- noop(v,l,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\ &\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)}) \end{aligned}$$

$$\begin{aligned} &getto(v,l2,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\ &\qquad r(c0,c1)^{(2)},r(c1,c0)^{(2)},r(c0,c0)^{(2)},r(c1,c1)^{(2)},at(t0,c0)^{(2)},at(t0,c1)^{(2)})\\ &:- getto(v,l1,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\ &\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)})\\ &\wedge drive(v,l1,l2,r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)},\\ &\qquad r(c0,c1)^{(2)},r(c1,c0)^{(2)},r(c0,c0)^{(2)},r(c1,c1)^{(2)},at(t0,c0)^{(2)},at(t0,c1)^{(2)}) \end{aligned}$$

- Horn clauses for the primitive tasks:

$$\begin{aligned} &drive(0,0,0,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\ &\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)})\\ &:- at(t0,c0)^{(0)} \wedge r(c0,c0)^{(0)} \wedge \neg at(t0,c0)^{(1)} \wedge at(t0,c0)^{(1)}\\ &\wedge r(c0,c1)^{(0)} = r(c0,c1)^{(1)} \wedge r(c1,c0)^{(0)} = r(c1,c0)^{(1)} \wedge r(c0,c0)^{(0)} = r(c0,c0)^{(1)} \wedge r(c1,c1)^{(0)} = r(c1,c1)^{(1)}\\ &\wedge at(t0,c1)^{(0)} = at(t0,c1)^{(1)} \end{aligned}$$

$$
\begin{aligned}
&drive(0,0,1,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\
&\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)})\\
&:- at(t0,c0)^{(0)} \wedge r(c0,c1)^{(0)} \wedge \neg at(t0,c0)^{(1)} \wedge at(t0,c1)^{(1)}\\
&\wedge r(c0,c1)^{(0)} = r(c0,c1)^{(1)} \wedge r(c1,c0)^{(0)} = r(c1,c0)^{(1)} \wedge r(c0,c0)^{(0)} = r(c0,c0)^{(1)} \wedge r(c1,c1)^{(0)} = r(c1,c1)^{(1)}
\end{aligned}
$$

$$
\begin{aligned}
&drive(0,1,0,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\
&\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)})\\
&:- at(t0,c1)^{(0)} \wedge r(c1,c0)^{(0)} \wedge \neg at(t0,c1)^{(1)} \wedge at(t0,c0)^{(1)}\\
&\wedge r(c0,c1)^{(0)} = r(c0,c1)^{(1)} \wedge r(c1,c0)^{(0)} = r(c1,c0)^{(1)} \wedge r(c0,c0)^{(0)} = r(c0,c0)^{(1)} \wedge r(c1,c1)^{(0)} = r(c1,c1)^{(1)}
\end{aligned}
$$

$$
\begin{aligned}
&drive(0,1,1,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\
&\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)})\\
&:- at(t0,c1)^{(0)} \wedge r(c1,c1)^{(0)} \wedge \neg at(t0,c1)^{(1)} \wedge at(t0,c1)^{(1)}\\
&\wedge r(c0,c1)^{(0)} = r(c0,c1)^{(1)} \wedge r(c1,c0)^{(0)} = r(c1,c0)^{(1)} \wedge r(c0,c0)^{(0)} = r(c0,c0)^{(1)} \wedge r(c1,c1)^{(0)} = r(c1,c1)^{(1)}\\
&\wedge at(t0,c0)^{(0)} = at(t0,c0)^{(1)}
\end{aligned}
$$

$$
\begin{aligned}
&noop(0,0,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\
&\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)})\\
&:- at(t0,c0)^{(0)}\\
&\wedge r(c0,c1)^{(0)} = r(c0,c1)^{(1)} \wedge r(c1,c0)^{(0)} = r(c1,c0)^{(1)} \wedge r(c0,c0)^{(0)} = r(c0,c0)^{(1)} \wedge r(c1,c1)^{(0)} = r(c1,c1)^{(1)}\\
&\wedge at(t0,c0)^{(0)} = at(t0,c0)^{(1)} \wedge at(t0,c1)^{(0)} = at(t0,c1)^{(1)}
\end{aligned}
$$

$$
\begin{aligned}
&noop(0,1,r(c0,c1)^{(0)},r(c1,c0)^{(0)},r(c0,c0)^{(0)},r(c1,c1)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},\\
&\qquad r(c0,c1)^{(1)},r(c1,c0)^{(1)},r(c0,c0)^{(1)},r(c1,c1)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)})\\
&:- at(t0,c1)^{(0)}\\
&\wedge r(c0,c1)^{(0)} = r(c0,c1)^{(1)} \wedge r(c1,c0)^{(0)} = r(c1,c0)^{(1)} \wedge r(c0,c0)^{(0)} = r(c0,c0)^{(1)} \wedge r(c1,c1)^{(0)} = r(c1,c1)^{(1)}\\
&\wedge at(t0,c0)^{(0)} = at(t0,c0)^{(1)} \wedge at(t0,c1)^{(0)} = at(t0,c1)^{(1)}
\end{aligned}
$$

- Clause for the initial state and initial task:

$$
\begin{aligned}
&\bot :-\\
&at(t0,c0)^{(0)} \wedge \neg at(t0,c1)^{(0)} \wedge r(c0,c1)^{(0)} \wedge r(c1,c0)^{(0)} \wedge \neg r(c0,c0)^{(0)} \wedge \neg r(c1,c1)^{(0)}\\
&\wedge getto(0,1,r(c0,c1)^{(0)},r(c1,c0)^{(0)},at(t0,c0)^{(0)},at(t0,c1)^{(0)},r(c0,c1)^{(1)},r(c1,c0)^{(1)},at(t0,c0)^{(1)},at(t0,c1)^{(1)}
\end{aligned}
$$

## 2.1 Optimisation for the static part of the world

If some facts about the world cannot change (they are not part of any effect of any action), we do not need to introduce a boolean variable for it and propagate it through all predicates. Instead with record its value defined by the initial state and substitute this constant value in all places where we previously needed the variable.

In our example, the facts about the roads are constant, so we can simplify our encoding using the values $r(c0, c0) \rightarrow \bot, r(c1, c1) \rightarrow \bot, r(c0, c1) \rightarrow \top, r(c1, c0) \rightarrow \top$.

- The boolean variables for all the ground instances of predicates:

$$at(t0, c0), at(t0, c1)$$

- Horn clauses for the abstract tasks:

$$\begin{aligned} &getto(v, l, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \\ &:- drive(v, il, l, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \end{aligned}$$

$$\begin{aligned} &getto(v, l, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \\ &:- noop(v, l, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \end{aligned}$$

$$\begin{aligned} &getto(v, l2, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(2)}, at(t0, c1)^{(2)}) \\ &:- getto(v, l1, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \\ &\wedge drive(v, l1, l2, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}, at(t0, c0)^{(2)}, at(t0, c1)^{(2)}) \end{aligned}$$

- Horn clauses for the primitive tasks:

$$\begin{aligned} &drive(0, 0, 0, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \\ &:- at(t0, c0)^{(0)} \wedge \bot \wedge \neg at(t0, c0)^{(1)} \wedge at(t0, c0)^{(1)} \\ &\wedge at(t0, c1)^{(0)} = at(t0, c1)^{(1)} \end{aligned}$$

$$\begin{aligned} &drive(0, 0, 1, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \\ &:- at(t0, c0)^{(0)} \wedge \top \wedge \neg at(t0, c0)^{(1)} \wedge at(t0, c1)^{(1)} \end{aligned}$$

$$\begin{aligned} &drive(0, 1, 0, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \\ &:- at(t0, c1)^{(0)} \wedge \top \wedge \neg at(t0, c1)^{(1)} \wedge at(t0, c0)^{(1)} \end{aligned}$$

$$\begin{aligned} &drive(0, 1, 1, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \\ &:- at(t0, c1)^{(0)} \wedge \bot \wedge \neg at(t0, c1)^{(1)} \wedge at(t0, c1)^{(1)} \\ &\wedge at(t0, c0)^{(0)} = at(t0, c0)^{(1)} \end{aligned}$$

$$
\begin{aligned}
&noop(0, 0, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \\
&:- at(t0, c0)^{(0)} \\
&\wedge at(t0, c0)^{(0)} = at(t0, c0)^{(1)} \wedge at(t0, c1)^{(0)} = at(t0, c1)^{(1)}
\end{aligned}
$$

$$
\begin{aligned}
&noop(0, 1, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}) \\
&:- at(t0, c1)^{(0)} \\
&\wedge at(t0, c0)^{(0)} = at(t0, c0)^{(1)} \wedge at(t0, c1)^{(0)} = at(t0, c1)^{(1)}
\end{aligned}
$$

- Clause for the initial state and initial task:

$$
\begin{aligned}
&\bot :- \\
&at(t0, c0)^{(0)} \wedge \neg at(t0, c1)^{(0)} \\
&\wedge getto(0, 1, at(t0, c0)^{(0)}, at(t0, c1)^{(0)}, at(t0, c0)^{(1)}, at(t0, c1)^{(1)}
\end{aligned}
$$